12-2016

Automatic Assessment of Environmental Hazards for Fall Prevention Using Smart-Cameras

Jeffrey Kutchka
University of Arkansas, Fayetteville

Automatic Assessment of Environmental Hazards for Fall Prevention Using Smart-Cameras

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering

by

Jeffrey Kutchka
University of Arkansas
Bachelor of Science in Computer Engineering, 2007

December 2016
University of Arkansas

This thesis is approved for recommendation to the Graduate Council

---

Prof. Christophe Bobda
Thesis Director

Prof. Jia Di
Committee member

Prof. John Gauch
Committee member

Abstract

As technology advances in the field of Computer Vision, new applications will emerge. One device that has emerged is the smart-camera, a camera attached to an embedded system that can perform routines a regular camera could not, such as object or event detection. In this thesis we describe a smart-camera system we designed, implemented, and evaluated for fall prevention monitoring of at-risk people while in bed, whether it be for a hospital patient, nursing home resident, or at home elderly resident. The camera will give a nurse or caregiver environmental awareness of the at-risk person and notify them when that person performs an action that could lead to a hazardous event. This camera uses Haar Cascade facial detection techniques, Histogram of Oriented Gradients(HOG) for person detection, and Mixture of Gaussians (MOG) background subtraction while operating. Regions are created by a person from a graphical user interface (GUI). The camera looks within these regions to find a face, a standing person, or just a change in the image. A notification is sent to the smartphone of the nurse or caregiver of the corresponding at-risk person when the camera finds one of these three detections in the drawn region. The Cloud is utilized to send the notification to the nurse or caregiver's smartphone. Given a properly placed camera and properly drawn regions, notifications can be sent when the at-risk person is doing an action that demands the attention of the nurse or caregiver, such as getting out of bed. The smart-camera does contain drawbacks. It is likely to give alerts when visitors are in the room, and it does not know how to pause notifications when a nurse, doctor, or caregiver comes into the room.

Dedication

To my family and friends for the company and support.

Acknowledgements

Thanks to all the professors who provided instruction and challenged me throughout my years as a student. Thanks to Dr. Bobda for his guidance and support as an advisor. Thanks to all the members of the Smart Embedded Systems Laboratory that helped with the project. Especially Nicolas Edwards for his work in putting the physical parts together along with his work on the power supply, and Joel Mandebi Mbongue for his work in making the smart-camera Cloud enabled. Thanks to Itseez for the OpenCV project, thanks to Trolltech for Qt, and thanks to the Raspbery Pi Foundation for the Raspberry Pi.

Table of Contents

List of Figures

List of Tables

1 Introduction

With embedded systems becoming faster and cheaper and advances being made in the field of Computer Vision, the potential for new applications has increased. One new device that has become more viable, the smart-camera, is a camera and an embedded system connected together that can perform computational routines a regular camera cannot. Examples of computational routines a smart-camera can do are object detection, background subtraction, and object tracking. An example application of a smart-camera is fall prevention in sleeping areas of people at risk for falling, such as hospital patients, nursing home residents, or elderly living at home. Our device is an example of that application.

1.1 Motivation

Falls in hospitals in the United States are dangerous and costly. It is estimated that 11,000 people die from falls in hospitals each year[4]. One study conducted on more than 88 million patient days found that 315,817 patients fell, 25% of which were injured, and 2% received a fracture[5]. Another study found on average 6.27 days were added to a patient's stay when injured by a fall while already in the hospital[6]. Patient falls resulting in injury cost on average $14,000[7].

Besides the monetary cost, the quality of life of the fall victims is impacted too. In the elderly aged 65 and older, "about one in ten falls results in serious injuries such as hip fracture, other fractures, subdural hematoma, or traumatic brain injury.[8]" Among older people, recovery from fall related injuries is often delayed which leads to difficulty in relearning the ability to walk[9]. Also, post-fall anxiety can occur among the elderly causing some fall victims to become overcautious which can lead to difficulty in relearning to walk[9]. Without proper walking, weakness will increase and a normal gait is unlikely to form quickly, both factors which can increase the odds of another fall from occurring[9].

1.2 Solution

To mitigate the problem of at-risk people of falling, a couple of research groups have created solutions. The University of Illinois created a get-up event detection system that de-

tects when a person gets out of bed. It processes video collected from a Kinect to monitor a person from a side view. Their product works well but it requires mounting a Kinect from the side of a person's bed. There is a chance that the device performing the computations is also in the room which could be an inconvenient obstacle if it is a large machine. Another solution is the University of Purdue's SmartGait monitoring system. This device works by monitoring a person's gait which can be an indicator that a fall is about to occur. It is useful for monitoring while an at-risk person is walking, but it does not help when a person is in bed. Also, neither of the solutions mentioned support notifying a nurse or caregiver.

Our solution to mitigate falls from occurring is a smart-camera. It is much like the University of Illinois's system, but is contained within a single camera housing that only requires a power cable and a wifi connection. It works by monitoring at-risk people while they are in bed. It detects events that potentially precede hazardous events, such as a person getting out of bed or a person waking up, both events which a nurse or caregiver should know about to prevent an at-risk person from falling. When it detects an event, it is able to alert a nurse or caregiver with smartphone notifications by utilizing the Cloud.

2 Chapter 2

2.1 Background

Falls in hospitals, nursing homes, and residences regarding at-risk people, especially the elderly, is a serious problem. We propose a smart-camera as a solution to mitigate this problem. Our smart-camera will provide nurses or caregivers with awareness about what the at-risk person is doing while resting in bed, and it will notify them in case an event occurs that could require assisting the at-risk person in walking, particularly in guiding the at-risk person to the restroom. Events can be getting out of bed or waking up. When an event occurs, the nurse or caregiver receives a notification on their smartphone which is generated by the smart-camera and sent via the Cloud.

2.1.1 Key Concepts

Our smart-camera system uses several Computer Vision techniques that each demand a description of their own. Histogram of Oriented Gradients (HOG), a method that trains on images of one type of object (car, person, bicycle, and others), is used to detect standing people, or at-risk people in this case. Support vector machines (SVM) are used for training with HOG. A Haar cascades classifier is used with a facial detection template to detect when an at-risk person is sitting up. For background subtraction the algorithm known as MOG2 from the Open Source Computer Vision (OpenCV) library is used. MOG2 is short for mixture of Gaussians 2. Aside from discussing Computer Vision techniques, a description of the Cloud as it is used with this project is also provided.

Histogram of Oriented Gradients

Histogram of Oriented Gradients was created by Navneet Dalal and Bill Triggs who are two researchers from the French National Institute for Research in Computer Science and Automation (INRIA)[10]. The first publication of HOG was in the 2005 Conference on Computer Vision and Pattern Recognition (CVPR)[10].
The HOG algorithm works as follows. It takes a 64x128 grayscale image as input. Color is not important in this version of HOG. Then the gradient vector (size and orientation) is

computed for each pixel[11]. The magnitude of the gradient of a pixel is calculated by taking the difference of the intensities of two horizontal neighbors and the difference of the intensities of the two vertical neighbors then calculating the L2 norm of these differences[11]. The angle is calculated using the arctan function[11].

Then, this new 64x128 matrix of gradients magnitudes and angles are divided into 8x8 cells[12]. The matrix can now be viewed as a set of 128 (8x16) cells. Each cell needs to have a 9-binned histogram of oriented gradients calculated for it[12]. This is done by creating a histogram with 9 bins in it. The bins represent the nine non-overlapping 20°ranges from 0°to 180°[12]. Each pixel in the cell adds its gradient magnitude to whichever bin its gradient angle falls in. The algorithm covers gradients with angles between 180°and 360°by using signed gradient magnitudes. That is, angles between 180°and 360°are converted to the corresponding angle that will be in the range -180°to 0°. One way of calculating the appropriate histogram bin is by using the floor operator on the quotient resulting from the division of the gradient angle by 20[13]. The result of this floor operation is the index to the bin that the magnitude should be added to. The 9-binned histogram for the cell is complete after adding each pixel's corresponding gradient magnitude to the appropriate bin of the histogram.

Now the 64x128 matrix is divided into blocks. Each square block is 2 cells in width and 2 cells in height. For convenience, the matrix can be thought of by its dimensions in cell units so it is 8 cells by 16 cells. The block distribution is described as having a 50% overlap. This means that starting at the top left position of the matrix, the first block covers the first four cells of the first two rows. The next block is moved 50% to the right, or one cell width over, of the previous block. Blocks are created this way until the edge of the matrix is met. The next block is moved 50% down from the first block's position of the previous row. The block after that is moved 50% to the right of the previous block. This pattern continues until the 64x128 matrix is saturated with blocks. With a matrix of size 8 by 16 in cell units, there will be 105 (7x15) blocks.

For each block, the 9-binned histogram for each of the 4 cells of the block are concatenated together forming a vector (or array) with 36 elements. Then this vector is normalized by dividing each of its 36 elements by its L2 norm (other normalization methds can be used). This normalization is done to account for changes in illumination and contrast[10]. The final HOG feature vector is the concatenation of each of the 105 blocks' normalized vectors, so the vector contains 3780 elements. This final vector is input into an SVM that

has been trained to detect an upright standing person.
Most images that HOG is applied to are not 64x128 in size. What is done for images of larger size is that a window of size 64x128 is moved across the image performing HOG detection each time it is moved. It moves across the image in a similar manner to how the blocks were created when processing the 64x128 image. Like the 50% overlap parameter when processing blocks, a parameter known as win_stride[14] determines how much the window moves across the image per iteration. A window larger than 64x128 can be used. When the window is larger than 64x128, the image under this larger window must be scaled down to 64x128 and then processed the same as a 64x128 window would. It is important to only use window sizes that might contain a person. Using extra window sizes will increase the chance of false positives occurring and increase computation time.

Support Vector Machine

The SVM is a supervised learning model. It takes a set of vectors as inputs. A class label is also supplied with each of the input vectors. Since SVMs are binary classifiers, the class label can either be 1 (positive) or -1 (negative)[15]. Using these input vector and label pairs, hyperplanes are constructed that separate as much of the positively labeled vectors from the negatively labeled vectors. A simple version of this hyperplane can be visualized as a line in a 2-dimensonal space that separates two sets of data by their class type. The hyperplane is like this but works on n-dimensional data. Also, the SVM that HOG uses is a linear SVM, meaning the positive and negative classes have to be linearly separable (or mostly).
For person detection using HOG, an SVM was trained using 1239 64x128 images of people standing upright in assorted orientations[16]. This is the positive set. For the negative set 12180 64x128 images of random content were used, none of which contained a person standing upright[16]. Each positive and negative image was processed with the HOG algorithm, resulting in a final HOG feature as discussed at the end of section 2.1.1. These final HOG features, or HOG descriptors[10], are input into an SVM along with their appropriate class labels. The SVM trains on this data creating the separating hyperplanes. Once the SVM is trained, it can classify HOG descriptors from 64x128 images as either person or not person.

Haar Cascades Classifier

The Haar cascade classifier method was introduced by Paul Viola and Michael Jones in 2001 in the paper [17]. It uses a set of convolution filters known as Haar features to extract features of the image. An example of some Haar features can be seen in figure 2.1. For each Haar feature in the figure, the sum of the pixels under the white area is subtracted from the sum of the pixels in the black area, resulting in a feature value. For face detection, Adaboost is used on these extracted features to create a classifier with a high success rate at classifying faces from non-faces[1]. When training for face detection, using Adaboost allows for the number of Haar features used to go from more than 160,000 to 6000. This originally high number is all possible sizes and locations of the Haar features in a 24x24 window [1]. A cascading process is used as a way to classify non-faces quicker without having to check all 6000 features.

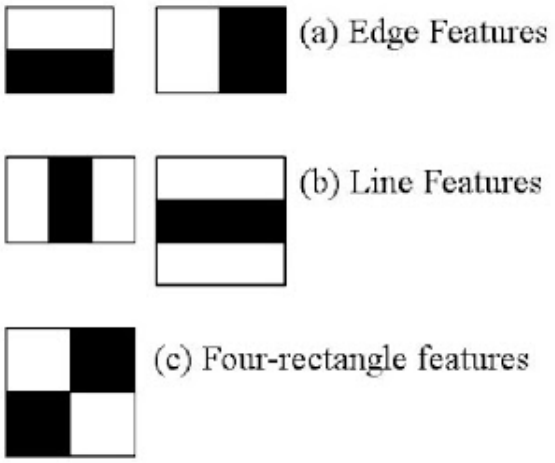

Figure 2.1: Some haar features[1]

Background Subtraction using Mixture of Gaussians

Our smart-camera performs background subtraction by using a Mixture of Gaussians technique. This technique is covered by its authors Zoran Zivkovic and Ferdinand van der Heijden in the 2006 article "Efficient adaptive density estimation per image pixel for the task of background subtraction[18]." It is known in the OpenCV library as MOG2.
One method of background subtraction, frame differencing, works by using one f rame as a background and comparing new frames against this background. This comparison is made

by taking the difference of the intensities of each pixel in the background and the corresponding pixel in the current frame under consideration. Pixels with corresponding differences that do not equal 0 are considered as foreground (A threshold can be set such that if the magnitude of the difference is greater than the threshold, the pixel is marked as foreground). This method is intolerant of illumination changes and shadows, and requires a background frame to be selected.
The MOG2 technique is more tolerant of illumination changes and is capable of identifying some shadows. Also, a background frame does not need to be set. It can determine background from foreground using its Gaussian mixture models. MOG2 uses a Gaussian mixture model on each pixel under consideration. It defaults to using 5 Gaussians per mixture model, with a history size of 500 elements.

The Cloud

Our smart-camera uses Google Cloud Messaging (GCM)[19]. The GCM service is a messaging system used to send notifications to smartphones. Its main feature is that it conserves battery power on smartphones. It does this by checking for all new notifications for a smartphone at regular intervals rather than having each individual app installed check at their own intervals. This allows the smartphone's radio to be off longer which conserves the battery. GCM is used by our smart-camera to send alerts to the smartphones of nurses or caregivers when an event is detected.
GCM works as follows. An app on a smartphone sends its sender ID, an ID unique to all copies of an app, to a GCM server for registration. The GCM server responds to the app with a unique registration ID. The app sends the registration ID to a web server which the smart-camera delivers alerts to. The web server stores the registration ID along with information related to that ID, such as a name. When a notification needs to be sent to a phone, the server determines the correct registration ID and sends that along with a message to the GCM server. The GCM server forwards the message to the correct smartphone.
An example of this in action is our smart-camera system. A caregiver installs the Android app that receives notifications. The app will register with the GCM server receiving a unique registration ID. The app will send this registration ID and a user login to a Glassfish web server. The web server is responsible for receiving alerts from the smart-

camera and getting the notifications to the appropriate caregivers' smartphones. When the smart-camera sends an alert to the web server, the web server determines which users (caregivers) should receive a notification and sends an alert along with the registration ID of each of these users to the GCM server. The GCM server sends the messages to the users' smartphones in the form of push notifications.

2.1.2 Related Work

Our smart-camera is not the first of its kind. Similar projects exist for fall detection and prevention that use smart-cameras. The University of Missouri created a fall detection system that uses a Microsoft Kinect to monitor patients in their hospital rooms[20]. The University of Illinois's Advanced Digital Sciences Center (ADSC) created a fall prevention system that also uses the Microsoft Kinect to monitor patients in their hospital rooms[21]. For fall prevention in any environment, the University of Purdue created a system the uses a smartphone and its camera to monitor the gait of an individual[22].

University of Missouri's "Rewind" system

The University of Missouri's "rewind" system uses a Kinect to monitor hospital patients while they are in bed. The Kinect is an RGB-D camera, meaning it collects RGB data like a typical camera and it also collects depth data. The Kinect is placed near the ceiling above a wall mounted flat screen television and faces the patient's bed at a downward angle. The system collects data on a computer that is hidden behind the television. Unfortunately, the inner workings of the algorithm the authors use are not publicly available. However it is known that this system detects falls by performing background subtraction on the depth channel. The system is always recording and when a fall is detected, the recording of the time around the event is stored. This is the "rewind" feature. This video allows caregivers to analyze what the scene was like around the time of the fall so that they may learn from it.

Data was collected over 100 weeks from 6 occupied patient rooms inside a hospital. They claimed their system had a 92% sensitivity rate and 95% specificty rate, and averaged 11 false alarms per month[20].

There are three important differences between the University of Missouri's "rewind" project and the smart-camera system. The "rewind" project is a fall detection system and not

a fall prevention system, so it is only good for detecting falls after they have happened. Also, it uses background subtraction on depth images and not RGB or grayscale images. Lastly, it uses a bulky system consisting of a Kinect, a EEEBox[23], which is a small form factor computer, and an external hard drive all inside the patient's room. The smart-camera system only takes up the space of a small camera housing, which is dome shaped with a 5" diameter and 3.3" height (see fig 2.2), and only needs to plug into a 12V source.

University of Illinois's Get-Up Event Detection System

The University of Illinois's ADSC created a similar system that detects when a patient gets up from bed. It is like the "rewind" system in that it uses the depth channel from a Kinect camera but also the RGB channels. The ADSC uses the Kinect such that it views the patient's bed from the side at a slightly elevated angle and at a distance such that all of the bed fits within the camera's view.

The ADSC system separates the data from the Kinect into 8 regions of interest that cover the patient in bed. For each region motion history images, HOG, and histogram of optic flows features are gathered for both the RGB data and the depth data, resulting in 48 features[21]. The system is trained using 40 positive video samples and 200 negative video samples each of which are 5 to 10 seconds long[21]. The authors claim a 98.76% recognition accuracy, and they found that the system only needs to use the motion history images to achieve high accuracies.

Our smart-camera system is different from the ADSC system in the following ways. They both use HOG, but ours is trained to detect a standing person, and ADSC's is trained to detect a person getting out of bed. ADSC's system does not utilize background subtraction techniques and ours does. Also, ADSC does not mention using routines from the OpenCV library, so it is assumed that it does not. Our system does use routines from the OpenCV library. Also, the ADSC system requires a different camera position than our smart-camera system.

University of Purdue's SmartGait Monitoring System

The University of Purdue created a preventative system that monitors the gait of a patient. It uses a smartphone attached to the waist of the person being monitored. The smartphone is altered so that its camera faces downward towards the feet. Single colored

markers roughly the size and shape of a silver dollar are placed on the shoes of the person. The placement of the marker is near the front and top part of the shoe above where the toes are. An application on the phone collects gait data by detecting these colored markers and making the appropriate measurements. It measures gait length, gait width, and walking speed. The SmartGait system has 95% accuracy when measuring step length and 90% when measuring step width.

The SmartGait system is useful for determining if a fall is about to occur based on gait data, and it can be used as an indicator to determine if there are other methods that need to be done to reduce the chance of a fall. The University of Purdue's news states:

> Health care professionals could use data from SmartGait to make an assessment and recommend fall-prevention measures such as exercise, physical therapy or vision correction. The device also might be worn over time to gauge a patient's progress in walking confidently[22].

SmartGait is less similar to our smart-camera system as compared to the University of Illinois's and the University of Missouri's systems. It's most important difference is that it is a mobile application that can monitor a patient anywhere they walk and therefore it is not limited to monitoring a patient while they are in their hospital bed. However, it is more invasive in that it requires the monitored person to wear a belt with a slightly protruding smartphone attached and it needs colored markers to be placed on each shoe of the monitored person.

2.2 Hardware Platform

Our smart-camera is an embedded system with a camera attached. It uses a Raspberry Pi 2 (RPI2) [24] as the embedded system, a SainSmart camera module to collect video data [25], an Edimax EW-7811Un 802.11n wifi adapter [26], and everything is enclosed inside a dome shaped camera housing of dimsensions 5" in diameter and 3.3" in height.

2.2.1 Raspberry Pi 2

The RPI2 is a low powered embedded system (<15 watts) that could be purchased for about $35 at its release date[27]. It is equipped with a gigabyte of RAM, and it uses a 900Mhz quad-core Advanced RISC Machine (ARM) Cortex-A7 CPU[24]. The RPI2 has 4 USB ports, an HDMI port, a 10/100Mbps Ethernet port [28], an audio jack (not used), a

camera interface specifications (CSI) camera interface, a display serial interface (DSI) display interface (not used), a Micro Secure Digital (SD) slot, and a VideoCore IV 3D graphics core[24].

2.2.2 SainSmart Camera Module

The SainSmart camera is a 5 megapixel camera that is capable of recording video in 1080P, 720P, and 640x480p60/90 video[25]. It uses the CSI interface and attaches to the RPI2 with a ribbon cable. It does not use a USB slot. It does not contain an infrared filter, so it can be used at night or in the dark when coupled with an infrared (IR) lamp. Our smart-camera uses 22 light-emitting diodes operating in the IR spectrum as a source of IR lighting. These LEDs are contained within the camera housing. They operate from a 12V power source while the RPI2 uses 5V, and this difference must be addressed to have a properly working system. To control the camera from software, the RaspiCam C++ API is used. This API makes the SainSmart camera module compatible with the OpenCV library.

2.2.3 Edimax Wifi Adapter

The Edimax EW-7811Un wifi adapter is a nano USB adapter. It is advertised with a max throughput of 150Mbps that is capable of operating with the 802.11b, 802.11g, or 802.11n standards. The EW-7811Un supports HostAP mode[29], a wireless mode that allows the RPI2 to act as a wireless hotspot. This feature is in the plans for possible future work with our smart-camera so that it can be configured without an Ethernet cable.

2.2.4 Camera Housing

The case our smart-camera uses is of an unknown manufacturer. It came from the merchant uxcell on amazon.com[30]. Its design allows for the option of LEDs to be added, which we have. It is dome shaped, and would fit snuggly in a 5.7” by 5.5” by 3.3” box. See figure 2.2 for an image of the camera housing. The RPI2 rests on the bottom inside the case. The camera is mounted near the top of the dome looking outward. See figure 2.3 for an image of the camera housing opened with the RPI2 and camera module inside.

Figure 2.2: The camera housing used to hold the smart-camera's RPI2 and camera module[2]

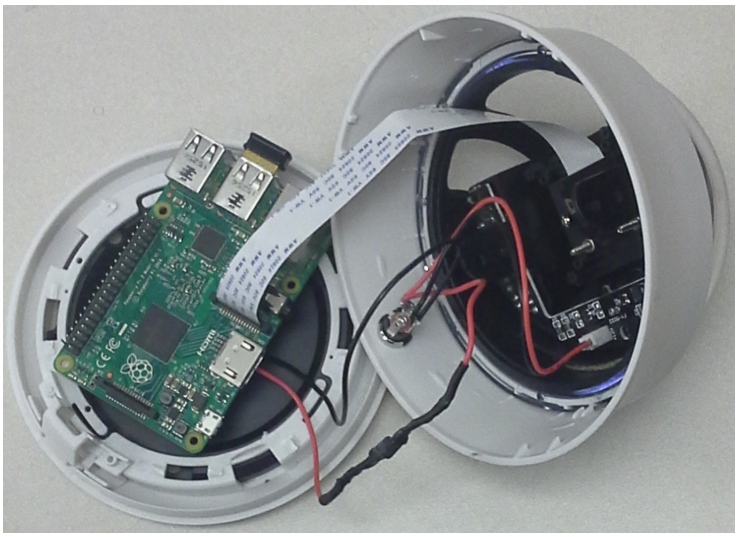

Figure 2.3: The camera housing with the case open. The RPI2 is visible. The camera module is in the part of the housing that is not holding the RPI2. It is on the end of the ribbon cable that is not inserted into the RPI2[2].

2.3 Software Platform

2.3.1 Raspbian Operating System

Our smart-camera operates on the Raspbian Jessie operating system[31]. Jessie is the title of the release version. Raspbian is a Debian based open sourced operating system that is designed to be run on Raspberry Pi boards. Software packages found in its packaging system are optimized to run on the Raspberry Pis' ARM processor. They make use of the Pis' hardware floating point units rather than using software to do floating point operations. Raspbian is a free operating system that uses the Linux kernel and is therefore covered by the general public license (GPL).

2.3.2 OpenCV

For Computer Vision operations, Itseez's Computer Vision library, OpenCV, is used[32]. OpenCV is an open source library that is licensed under the Berkeley Software Distribution (BSD) license meaning it can be used in academic, commercial, and other settings without fee. OpenCV

> "...has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.[33]"

Because of issues with the MOG2 algorithm in OpenCV 2.4.9, OpenCV version 3.0.0 is used with the smart-camera. The C++ application programming interface (API) is used. The HOG person detector, Haar Cascade face detector, and the MOG2 background subtraction algorithm are all contained within the OpenCV library.

2.3.3 Qt

Our smart-camera system needed a graphical user interface (GUI) for selecting regions of custom shape. OpenCV could not be used for this. Because of its cross-platform ability, the Qt (pronounced "cute") framework was chosen for this job. Qt is cross-platform,

meaning GUIs built with Qt that do not use operating system specific code will work in Microsoft Windows, Apple's Mac OS X and iOS, Linux, Android, and some lesser known operating systems[34]. Qt is available with commercial and non-commercial licenses depending on the intent of the target application[35]. Because of our academic purposes, we are using it with the GPL. Our GUI uses Qt version 5.2.1.

2.3.4 LibCurl

Our smart-camera system requires communication with a server over the internet to alert caregivers. To do this, the libcurl library is used. Libcurl is an open source C library licensed under a derivative of the MIT/X license[36]. It is free to use in anyway, including commercial products[36]. Libcurl provides support for

> "...client-side URL transfer library, supporting DICT, FILE, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, Telnet and TFTP. libcurl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password authentication (Basic, Digest, NTLM, Negotiate, Kerberos), file transfer resume, http proxy tunneling and more[37]!"

Specifically, our system uses it for Secure Copy, or scp, and HTTP POST. Cross-platform support is provided, but support is only needed for Linux. Our system uses version 7.35 built with scp support.

2.3.5 LibConfig

To provide support for configuration files so that hard-coding of parameters directly into the source code does not have to be done, the libary LibConfig is used with our smart-camera system. Configuration files are structured text files. See list 3.2 for an example. LibConfig provides C and C++ API bindings, meaning its API can be used in C or C++. It has cross-platform support like Qt, but only Linux support is needed[38]. Version 1.4.9 is used. It also uses the GPL.

3 Chapter 3

3.1 Implementation Details

Our smart-camera software can be partitioned into 5 parts. A text menu user interface is used to configure rectangular regions of interest. A GUI is used to mark regions of interest that are of custom shape. Another part, frame processing is where individual frames from the camera are checked to see if an alert is needed. The "settings.cfg" file is where many of the smart-camera's configuration settings are stored. Last is a section containing some details about sending notifications through the Cloud to the caregivers.

3.1.1 Text Menu User Interface

The text menu user interface is used to configure regions of interest before the camera begins processing. It only supports rectangular regions. They can be loaded from a file or they can be manually drawn using a mouse. To run the text menu user interface on the smart-camera requires opening the camera housing, connecting a keyboard and mouse to the RPI2, and connecting the RPI2 to a monitor, or a user can use secure shell (ssh) with X11 forwarding (the -X extension) to run the application remotely. The first method is a better experience but requires some disassembly, and the second method is easier in that it does not require disassembly, but the application's window updates very slowly when using ssh with X11 forwarding.
The interface can be seen in figures 3.1 and 3.2. At the top level, a user is given the option to add regions of certain types, such as standing, sitting-up, or lying down (lying down is not supported yet, but was put into the menu for future support); saving to a configuration file, windows.cfg, which can be seen in list 3.1; loading from a configuration file (windows.cfg only); begin processing video; or quitting.

Drawing Regions

To add a standing detection region, a user selects '1' from the top level of the menu (see fig. 3.1). Then the user draws a rectangular region where they would like the camera to search for a person standing up, as in figure 3.2. The steps to add a sitting-up detection

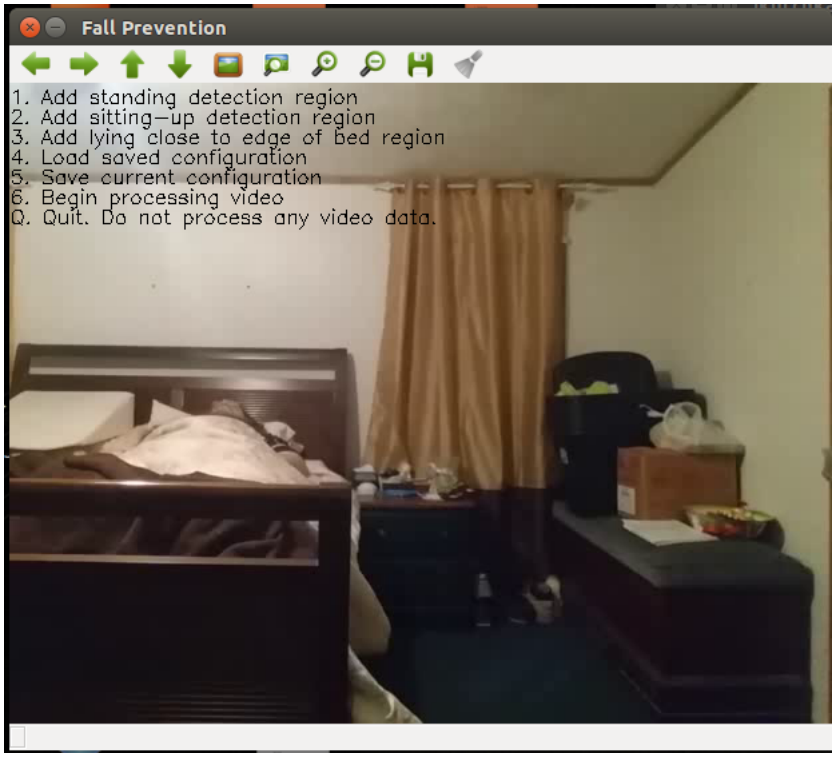

Figure 3.1: The top level of the menu interface.

region are the same as adding a standing detection region except the user selects '2' from the top level of the menu. The reason for separate region selection types is that they use different algorithms when MOG2 is turned off, and the caregivers receive different alerts depending on the region type. When MOG2 support is off, standing regions are processed using the HOG person detector, and sitting-up regions are processed using the Haar cascades classifier for facial detection. The smart-camera system supports as many regions as can be held in memory, but it is most likely that a user will run out of places to draw them before running out of memory.

Loading and Saving

After adding regions, a user can save them to a file, windows.cfg, with the '5' option. This file does not have a limit on how many regions it can store, but currently it only supports rectangular regions. An example windows.cfg file can be seen in list 3.1. Each region entry in this file contains an action type, X and Y coordinates to the top left corner of the region's rectangle, and width and height of the region's rectangle. The action type is used to

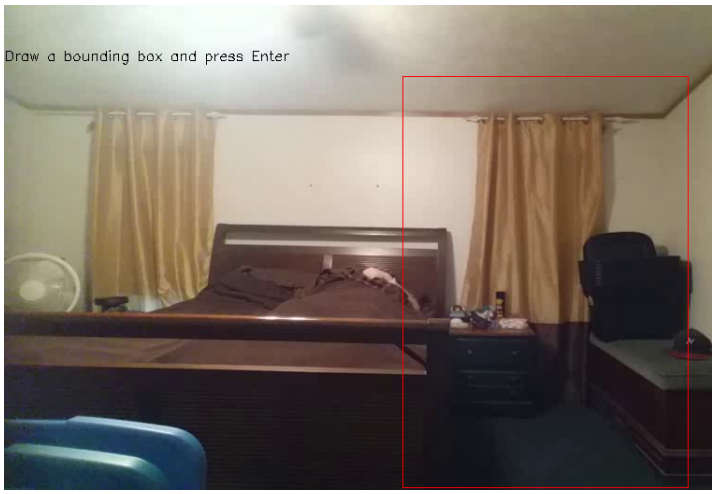

Figure 3.2: An example placement of a standing region. The region is enclosed in a red rectangle.

signify if the region is a standing or sitting-up region.

Listing 3.1: A sample windows.cfg file

```
Regions = (
  {
    ActionType = 1;
    Width = 419;
    Height = 155;
    LocationX = 200;
    LocationY = 59;
  },
  {
    ActionType = 2;
    Width = 223;
    Height = 431;
    LocationX = 31;
    LocationY = 31;
  } );
```

If the user chooses to use previously specified regions with the smart-camera, they can load them from the windows.cfg file using the '4' option. If the user has already created

some regions, then this load operation will erase what they have already created. However, the user can continue to add regions after the windows.cfg file has been loaded, which will not delete any of the regions already added by the file load operation.

Processing and Quitting

To begin processing with the smart-camera, the user can select '6'. The smart-camera will now look for faces in sitting-up regions and standing people in the standing regions, of if MOG2 is enabled, the smart-camera will look for a certain amount of foreground pixels in these regions. Also, if the user wishes to quit the application at any time, even after processing begins, they can hit 'q' or 'Q' on the keyboard.

3.1.2 Graphical User Interface

There is support for regions of custom shape if a user wants a region that is not rectangular. As of now these regions are only supported with the MOG2 algorithm. Our standalone GUI application (written in Qt) must be run to create these regions. Once started, 'L' must be pressed to load an image that can be used as a background reference when drawing these regions. The image needs to be the same resolution as what the smart-camera is configured to take video at, and it should probably be an image from the target scene. Once a background image is set, the mouse is used to draw regions. There is no limit on how many regions can be drawn. Once the user is done drawing regions, pressing 'S' will bring up a save dialog allowing the user to save their regions to a file. If the file is saved as "CareZones.bin" in the same directory as the smart-camera's executable file, "FallPrev", and if the "skipGUI" (this actually skips the text menu UI and begins processing) flag is enabled in the "setting.cfg" file, the user's custom drawn regions will be loaded into the smart-camera.
Figure 3.3 shows a region that has been created of a bed. The non-rectangular, shaded area surrounding most of the bed is the region. Figure 3.4 shows the resulting mask of that region. Another region and mask pair can be seen in figures 3.5 and 3.6 which show a region of a hat and the resulting mask. The masks designate which pixels are processed. The white pixels are processed with MOG2 and the black pixels are ignored.

Figure 3.3: An image of a bed with the outline of a custom region drawn around the border of the bed. The shaded area is the region.

Figure 3.4: The binary mask for the region in figure 3.3.

Figure 3.5: A custom region drawn around a hat. The custom region is shaded.

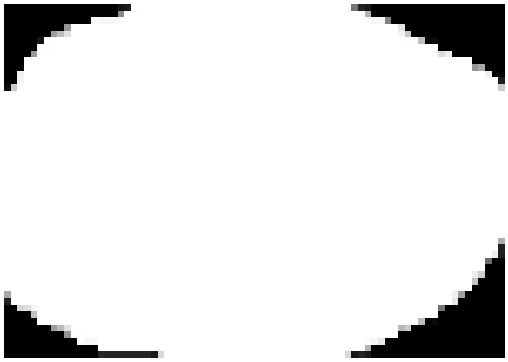

Figure 3.6: The binary mask for the shaded region in figure 3.6. The mask is only as big as it needs to be to cover the shaded region. The picture is scaled causing it to appear larger than it does in figure 3.6.

3.1.3 Frame Processing

Frame processing is the section of the smart-camera software where individual frames being read from the camera are processed to detect objects or events that require alerts to be sent to caregivers. Processing can be divided into three sections: using HOG, using Haar cascade classifiers, and using MOG2 for detections.

Figure 3.7 displays a flow chart showing the smart-camera's processing steps when using HOG and Haar. The 120 seconds referenced in the diamond step is a cooldown period that can be configured in "settings.cfg." It is worth mentioning that the bottom right step ultimately leads to a notification being received on a nurse or caregiver's phone even though the smart-camera is only responsible for getting the notification message to the GlassFish server.

Figure 3.8 displays a flow chart showing the smart-camera's processing steps when MOG2 is used. The range displayed in the top right diamond is configurable from the "settings.cfg" file. This flow chart is of a sitting-up region, but a standing region is similar only differing by the range of allowable areas.

Detecting with Histogram of Oriented Gradients

HOG can be used whenever a user configures the smart-camera to use standing regions, as described in the "Text Menu User Interface" section. The parameter "useMOG2" must be

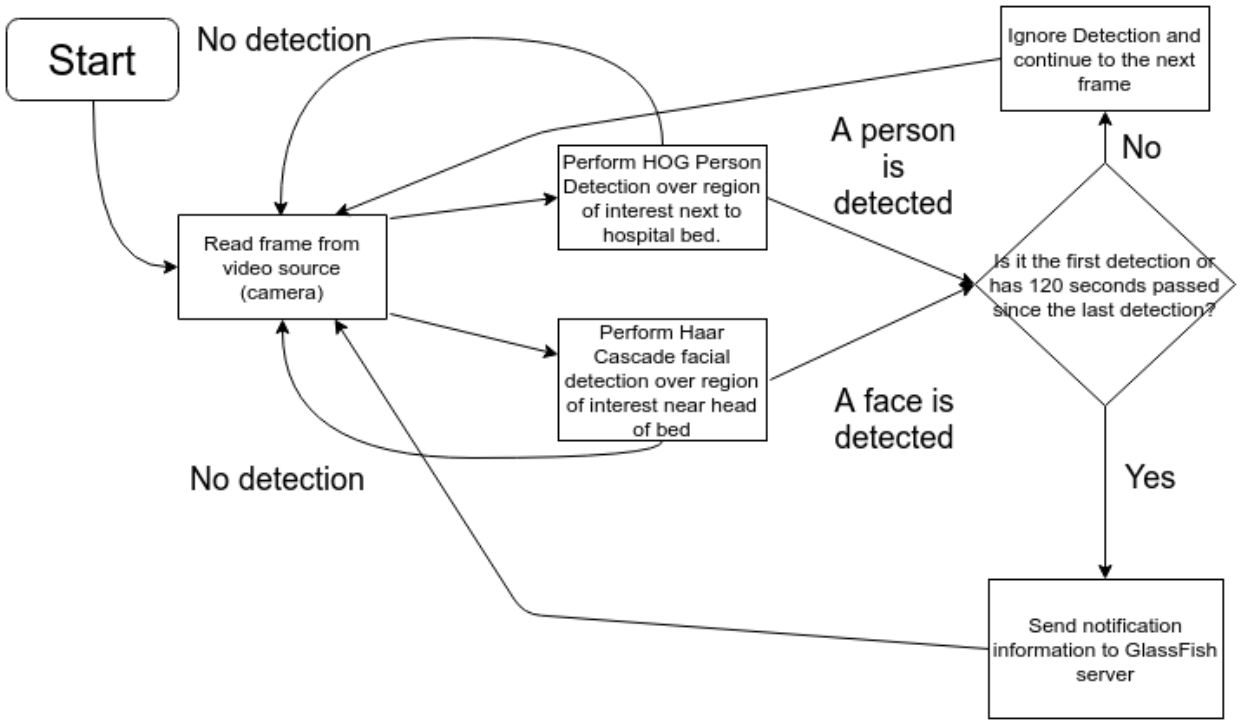

Figure 3.7: A high level overview of the smart-camera's processing steps when using HOG and Haar algorithms.

set to false in the "settings.cfg" file.

When a frame is read from the camera a matrix object, known as a Mat in OpenCV, is created for each standing region. The Mat for each region contains the part of the image that lies within the region's specified coordinates. Each Mat is processsed with the HOG algorithm. The line of code to do this is:

```
hog->detectMultiScale(roi_scaled, found, 0, Size(8,8),
                      Size(32,32), 1.05, 2);
```

The important parts of this line of code are that roi (short for region of interest) is one of the Mats mentioned above, found is a vector containing rectangles that contain a match within roi, Size(8,8) is the window stride that determins how much the 64x128 window moves across the region, and 1.05 is the base used when scaling up the detection window. If

```
found.size()
```

does not return 0, it is known that a detection has been made. When this happens, a check is performed to see if enough time has passed since the previous detection. If so, a

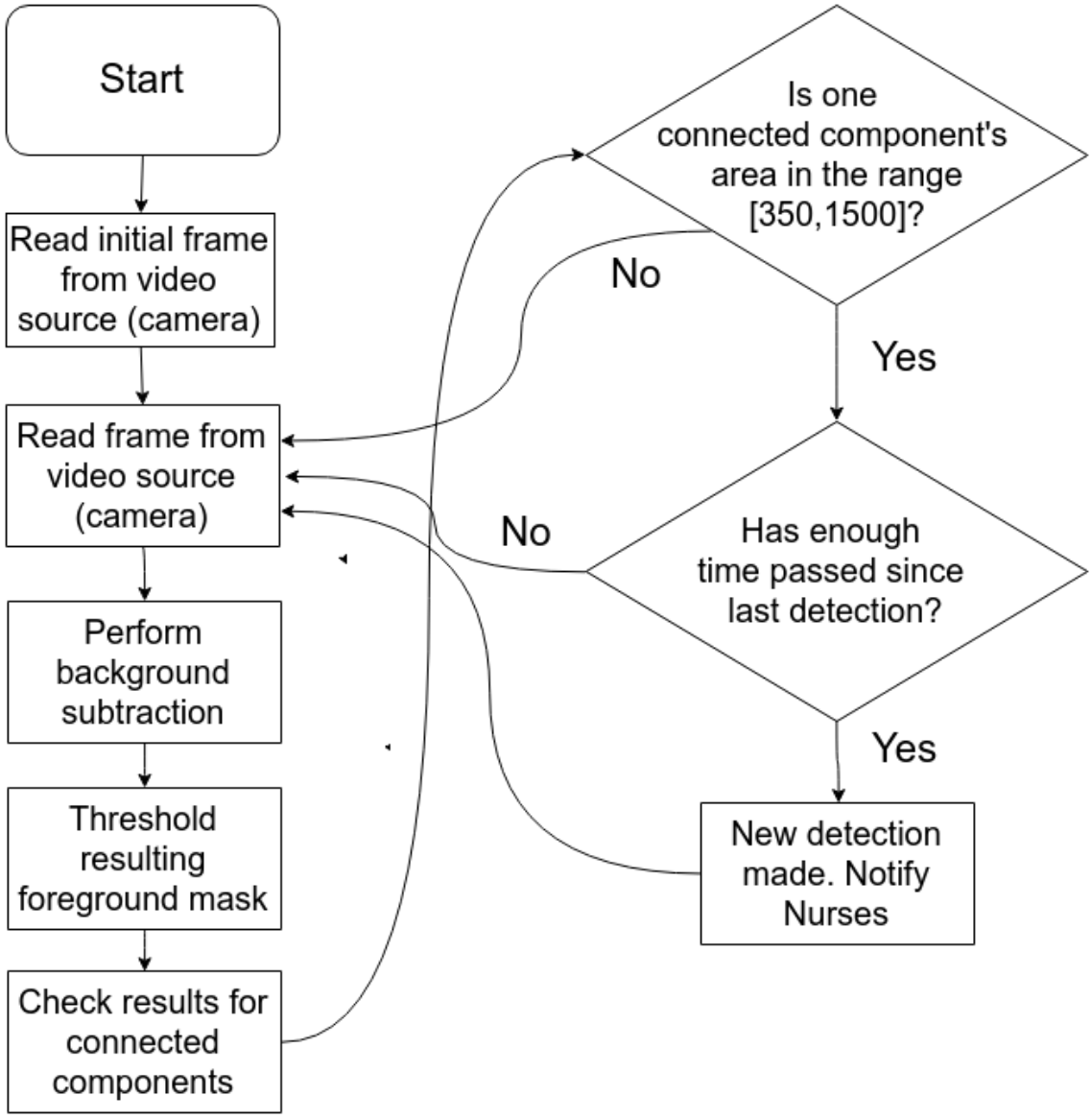

Figure 3.8: A high level overview of the smart-camera's processing steps when using the MOG2 algorithm for sitting-up detection. A diagram for standing detection would look the same but have a different specified range in the top right diamond.

notification is sent to the caregivers.

The base parameter in the detectMultiScale line of code is not part of the original HOG algorithm. It is used by OpenCV's implementation as a parameter used to scale up the 64x128 window. When the HOG algorithm finishes its search across a region with the 64x128 window, the search is performed again with a larger window. The dimensions of windows are computed by the following equations:

$$n = 1.05^{(p-1)} \tag{3.1}$$

$$x_{dim} = n * 64 \tag{3.2}$$

$$y_{dim} = n * 128 \tag{3.3}$$

where p is the current window iteration. The window dimensions after the first iteration are calculated with p equal to 2. After that window is done, another search is done with a new window size calculated with p equal to 3. This is repeated until the window size exceeds the region size, or until p equal to 30 (the GPU version of OpenCV's HOG algorithm allows the number of iterations to be tuned).

Detecting with Haar Cascade Classifiers

For regions that have been designated as sitting-up, and while "useMOG2" is false, as frames are read in from the camera, a 30 pixel by 30 pixel window is used to search for faces using a Haar cascades classifier template trained on detecting faces. This method is based on an exhaustive search approach, similar to how HOG sweeps it's window across a region of interest, of each of the regions, which tests the presence of the object (face) in the window at all positions and at different scales. The window is moved throughout the sitting-up regions looking for faces. If a face is found, a check to see if enough time has past since the last detection was made is performed. It then sends out notifications to caregivers.

To see an example of where to place a sitting-up region, refer to figure 3.9. The purpose of searching for a face in a sitting-up region has to do with an at-risk person's position while sleeping and while awake. It is assumed that a person will lie flat on their back with their eyes closed while asleep. In this position they will not be detected and no alerts will be made. When they awake, it is assumed they will sit up with their eyes open. In this position their face will be detected and an alert notifying the caregivers that an at-risk

person is sitting up will be generated. Also, if they are just waking up they will likely need to use the restroom, so when a caregiver receives the alert they can come to the aid of the at-risk person. This is important because many falls happen while the at-risk person is transitioning to the restroom or coming back from it.

There are some issues that need to be addressed with this detection method. One issue is that alerts will continually be made if an at-risk person such as a hospital patient is sitting up in bed throughout the day. A possible solution to this problem is to give the caregiver easy access to enable or disable the sitting up detection. When the at-risk person is down for sleep the caregiver can enable the detector, and when that person is awake for the day the caregiver can disable it. Another issue that might arise if the at-risk person is a patient is that their face may be covered with bandages. If this happens then the Haar cascade classifier will not detect a face. The patient will be incompatible with this sitting-up detection method.

Detecting with Mixture of Gaussians 2

MOG2 uses background subtraction as a way to detect movement in a region. The MOG2 algorithm can be used with either sitting-up or standing regions. It requires the setting "useMOG2" to be set to "true" in the "settings.cfg" file. When this is done, HOG and Haar methods are disabled. There is not a technical reason for MOG2 support to require HOG and Haar to be disabled. The software just has not been implemented to support all algorithms at once yet.

For sitting-up regions, like in figure 3.9, each frame is processed using the mixture of Gaussians algorithm found here [18]. When the algorithm is applied to a region of interest, the output is a Mat object. In this Mat object foreground pixels are given a value of 255 (white), shadows are given a value of 127 (gray), and everything else is given a value of 0 (black). The smart-camera is not interested in detecting shadows though, so a binary thresholding function is used to set all pixels within the Mat object that are not 255 (foreground) to 0 (background). Then, test the output Mat for an 8-way connected component that has an area within the region type's range. For sitting-up this range defaults to [350,1500]. The minimum and maximum are both tunable parameters, "FaceDetect_Min_Area_Of_Change" and "FaceDetect_Max_Area_Of_Change" respectively, that can be found in "settings.cfg." When a connected component is found with an area within this range,

a check is done to see if enough time has passed since the previous detection was made. If enough time has passed, a notification is sent to the the caregiver letting them know that a detection has been made in a sitting-up region.
Using MOG2 with standing regions is almost identical to sitting-up regions. The only difference is the range that is used, and the type of notification generated for the caregiver. Separate parameters for the minimum and maximum area of an acceptable connected component are available for standing regions. The parameters "PersonDetect_Min_Area_Of_Change" and "PersonDetect_Max_Area_Of_Change" are used for the minimum and maximum of this range. The default range is [1350,25000]. Also the notification sent to the caregiver will let them know that a detection has been made in a standing region.

Analysis

Background subtraction provides another method for detecting if an at-risk person is sitting up or standing. When using a sitting-up region similar to figure 3.9, a notification will be made when the at-risk person sits up in bed, much like the Haar method. The Haar method will have an advantage over MOG2 when a person is sleeping in an already angled position where they may not actually sit-up when they wake. When in this position, eye detection can be used in a region of interest rather than facial detection. An alert can be made when the eyes have been detected for a long enough period. If MOG2 is used in this position, false positives may result if the monitored person turns his or her head.
When using a standing region similar to figure 3.10, a notification will be made when the at-risk person exits the bed on the side with the region drawn, similar to the HOG method. In our tests, the MOG2 method would create an alert before a person completely entered the region of interest. This was not a problem and resulted in quick detection times. The HOG method took longer in each test and, even though a person was in the process of moving to a standing position, most often generated notifications from false positives of the HOG algorithm. That is, the HOG algorithm detected perhaps a part of a shirt or a section of an arm as a standing person and generated an alert from that rather than detecting a whole standing person. HOG does work though not as intended.
All of these methods except Haar are susceptible to false positives. The MOG2 methods have some protection against this in that they have a maximum connected component

Figure 3.9: An example of where to place a sitting-up region. The region is enclosed in the red rectangle. It is above the person lying in the bed.

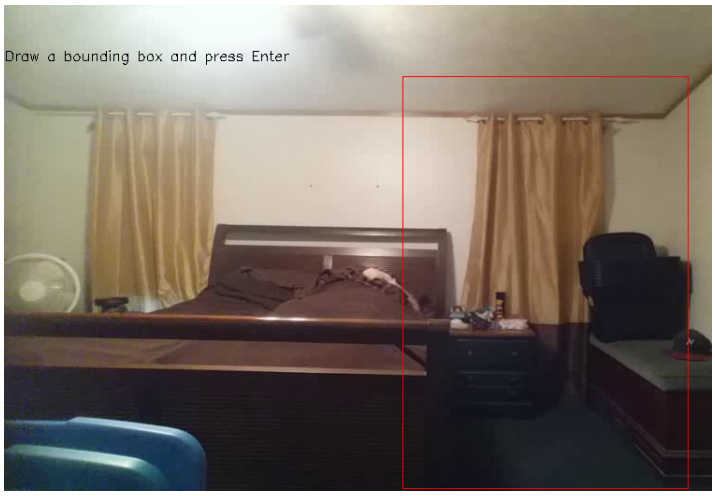

Figure 3.10: An example placement of a standing region. The region is enclosed in a red rectangle.

size. This will prevent false positives from being generated when someone obstructs the camera or if the camera is bumped or shakes. All of these events would cause a huge foreground with a connected component area roughly the size of the entire region of interest. For standing regions created with MOG2, anyone entering the region who is not the at-risk person will generate a false positive. This could be a caregiver coming into check on the monitored person or a relative giving the person a hug. Standing regions using HOG will generate false positives whenever a person other than the at-risk person enters them. In this environment the smart-camera would benefit by disabling its detecting abilities when a doctor, nurse, caregiver, or visitor entered the room.

3.1.4 Settings.cfg

Configuration parameters for the smart-camera are contained in the file "settings.cfg." An example of a settings.cfg file is seen in listing 3.2. The configuration file is separated into 5 parts: MOG2, SSL, HOG, FallPrev, and Debug.

MOG2 contains parameters that pertain to using the MOG2 algorithm. The "useMOG2" parameter is boolean. When it is true the MOG2 algorithm is used. When it is false the HOG and Haar algorithms are used. As discussed in section 3.1.3 "Mixture of Gaussians 2," the two FaceDetect parameters set the minimum and maximum acceptable area sizes in pixels that a connected component can have to create a notification when using a sitting-up region. The two PersonDetect parameters do the same except they are for standing regions.

The Secure Socket Layer (SSL) section contains information used by the smart-camera to perform a Secure Copy (SCP) to an image server. This secure copy mechanism requires a public and private Rivest-Shamir-Adleman (RSA) key. SCP is used from libcurl to store images onto a server if the feature is enabled. The "Public_Key" parameter stores the location of the public RSA key, and the "Private_Key" stores the location of the private RSA key.

The HOG section contains information used when a standing region is being processed by HOG. The "initial_power" parameter sets the initial value of p in the equation for n. See section 3.1.3 "Detecting with Histogram of Oriented Gradients" for the equation for n. This is useful when searching for a person in a window much larger than 64x128. Processing time is reduced by skipping smaller scales, and a false positive is less likely to occur

when skipping scales that are not important. The detectMultiScale function in OpenCV that performs the HOG algorithm does not provide an option to set p, so to get similar results the standing region must be shrunk by the factor n which is computed using the value p[39]. The "Number_Of_Scales" parameter is used only when GPU acceleration is available, which it is not with the RPI2. It sets a limit on how many times p can be incremented.

The FallPrev section contains parameters that pertain to the rest of the software on the smart-camera. The "Face" and "Person" cooldown parameters allow a user to set how many seconds must pass between notifications for sitting-up or standing respectively. The "Video_Frame_Width" and "Video_Frame_Height" parameters allow the user to set the resolution the smart-camera uses when taking video. The parameter "skipGUI" is used to turn off the text menu user interface. When it is true, regions are loaded from the file that the parameter "RegionSaveFile" is set to. "FPSconsole" is a boolean. When it is true the frame rate is displayed in the terminal. "RegionSaveFile" contains the file name to a binary file that contains custom drawn regions. The smart-camera loads regions from this file when "skipGUI" is true. If it is Null then windows.cfg will be used.

The Debug section contains parameters that were useful when debugging problems during development. "Cloud_Enable" allows notifications to be sent to caregivers over the cloud when set. When "Upload_Image" is set to true, an image will be sent to an image server. "Demo" is set to true when giving a demonstration with the smart-camera. It enables notifications through the cloud and disables extraneous windows from being generated while the software is running. When "RegionDebug" is true, a window is created that shows the video passing through a custom drawn region. This was helpful when creating this functionality.

Listing 3.2: Configuration parameters of the smart-camera

```
MOG2:
{
    useMOG2 = true;
    FaceDetect_Min_Area_Of_Change = 350;
    FaceDetect_Max_Area_Of_Change = 1500;
    PersonDetect_Min_Area_Of_Change = 1350;
    PersonDetect_Max_Area_Of_Change = 25000;
```

```
};
SSL:
{
    Public_Key = "id_rsa.pub";
    Private_Key = "id_rsa";
};
HOG:
{
    initial_power = 14;
    Number_Of_Scales = 30;
};
FallPrev:
{
    Face_Cooldown = 10;
    Person_Cooldown = 10;
    Video_Frame_Width = 640;
    Video_Frame_Height = 480;
    skipGUI = true;
    FPSconsole = true;
    PrivacySafe = true;
    RegionSaveFile = "CareZones.bin";
};
Debug:
{
    Cloud_Enable = false;
    Upload_Image = false;
    Demo = false;
    RegionDebug = true;
}
```

3.1.5 Uploading Images and Sending Notifications through the Cloud

The smart-camera has some capability in place to upload images to a server should the feature be needed. The images are uploaded to a server when a detection is made. To do this, the support for the SCP protocol found in libCurl is used. A Universal Resource Locator (URL) is created containing the server address, a file path, and a file name. An example URL looks like (without line breaks):

```
scp://fallprev.csce.uark.edu/var/www/html/images/
FallPrev20160709-055325PM.png
```

Here "fallprev.csce.uark.edu" is the server address. "/var/www/html/images/" is the file path where the image will be stored on the server. "FallPrev20160709-055325PM.png" is the file name that will be used to identify the image on the server. LibCurl is used to upload the image file through SCP. The RSA keys mentioned in the "settings.cfg" section are used when performing the SCP upload. After the image transfer, the image file name can be used to refer to the image in the notification URL mentioned below.

The smart-camera system uses the Google Cloud Messaging (GCM) [19] to send notifications to caregivers when detections are made. The smart-camera does this by communicating an action type (which is currently sitting-up or standing) and a room number to an Oracle GlassFish server version 4.0 [2]. The GlassFish server sends this information through the Cloud to the appropriate caregivers who receive the information on their Android smartphones.

The smart-camera uses HTTP POST to communicate with the GlassFish server. A URL is formed in 2 parts. The first part is a server address, port number, and a path to the notifyUser application on the GlassFish server. For our purposes it always looks like (without line breaks):

```
http://fallprev.csce.uark.edu:8080/healthCare.Restfull
/resources/userService/notifyUser
```

where "fallprev.csce.uark.edu" is the server address, "8080" is the port, and "/healthCare.Restfull/resources/userService/notifyUser" is the path to the application. The next part of the URL contains information pertinent to the notification. It is:

```
?roomID=1&imageName=FallPrev20160709-055325PM.png&actionType=1
```

There are three variables in this part: “roomID” which is the room number the at-risk person is in if at a hospital or nursing home, “imageName” which is the file name of the image, and “actionType” which signifies whether the alert is for sitting-up or standing (the 1 is for sitting-up. 2 is the number used for a standing alert). This URL is formed as a string object from the c++ standard library [40]. After combining all the parts of the URL, it is sent to the GlassFish server using the function call:

```
curl_easy_setopt(curl, CURLOPT_URL, fullURL.c_str());
```

from the libCurl library. The full URL looks like:

```
http://fallprev.csce.uark.edu:8080/healthCare.Restfull/
resources/userService/notifyUser?roomID=1
&imageName=FallPrev20160709-055325PM.png
&actionType=1
```

without the line breaks.

4 Methodology, Results, and Analysis

4.1 Methodology

A testing environment inside a hospital or nursing home with real at-risk people is needed to get good data on the smart-camera, such as a false positive rate when nurses or caregivers are in the at-risk person's room. Resources are not yet available to collect data in this kind of environment. However, we were able to take some measurements using data collected from other sources. The amount of time between when a notification url is received by the GlassFish server to when a notification is received on an Android phone was measured. Data about the duration between when a monitored person begins moving to when a detection is made by the smart-camera was also taken. Also, the frame rate the smart-camera operates at was measured for each algorithm.

4.1.1 Cloud Delay

To measure the time it takes for a notification to travel across the cloud once the GlassFish server receives a notification url, two timestamps were taken. The first timestamp happened when HTTP POST requests were sent from the machine the GlassFish server operated on to the GlassFish server. The second timestamp was taken when a notification arrived on the smartphone. The duration between these two timestamps was taken as an estimate (because it does not account for how long it took the server to contact itself) of the amount of time it takes a notification to reach its destination Android smartphone over the cloud. Also, the smartphone was using the cellular network (AT&T 4G LTE) and not wifi. The smartphone's cellular radio was active and not in a power saving mode.

4.1.2 Detection Times

To measure the time it takes for a monitored person to be detected once they begin to sit up or get out of bed, a video was made of a person getting out of bed 10 times. Before getting up, the person would lie in bed without movement. The video was cut at approximately the first sign of motion that was seen (by a person) for each of the 10 times the person got up, and cut again well after the first detection was made but before the

next attempt occurred. These are the clips used for measuring sitting-up times. When the video was made, the person would pause briefly after sitting up. Another 10 videos were made by cutting the video at the first motion towards standing up from this sitting-up position until the person had already stood up and left the video. These clips were used to measure detection time of standing up regions. To measure the detection times, two "windows.cfg" files were made, "A" and "B". "A" contained a sitting-up region that was drawn as seen in figure 3.9 where the region lies on the headboard above the lying person's body. "B" contained a standing region drawn next to the bed as seen in figure 3.10 where the region is next to the bed. To measure detection time for sitting-up regions, the region from file "A" was used. Each of the 10 clips cut from the original video were input into the smart-camera software and processed as video files rather than live video from the SmartSain camera module. A timestamp was taken at the beginning of video processing and another timestamp was taken when the first detection was made. The duration between these two timestamps was used as the detection time for sitting-up regions and whichever algorithm is being used. This process is done once for Haar cascades and once for MOG2. The 10 clips cut to measure standing-up detection times are run through the smart-camera software again but using file "B". Using the same process, detection times were measured for standing regions and the HOG and MOG2 algorithms. Also, the video clips are 616x480 and encoded with Xvid.

4.1.3 Frame Rate

To measure the frame rate that the smart-camera runs at, live video was taken of image content that did not change. The camera used the 1280x720p resolution. A timestamp is taken immediately before the first frame and immediately after the 500th frame of the live video. The frame rate is measured as the duration between the two timestamps divided by the number of frames that have passed ,500, in the camera feed. This process is done for each algorithm of each region type. Measurements are made using the same region for each region type when repeated with the other algorithm for that region type.

4.2 Results and Analysis

Using 20 test cases, we measured the average delay when using GCM as 3.27 seconds[2]. Every notification was received. These results are in order with a more thorough study of

	Experiments			
	Offline		Online	
	WiFi	Cell	WiFi	Cell
1st Quartile	4	5	1	2
3rd Quartile	11724	1759	4	6
Median	2449	8	2	4
Average	5594	328	2	3

Table 4.1: A table from [3]. Offline times are measured by sending out notifications at random times. Online times are measured when a user is actively using their phone.

GCM conducted in 2014, which measured the mean notification delay to be 3 seconds for actively used devices on a cellular network[3]. The results from that study can be seen in table 4.1. In this table, offline refers to measuring delay of random notifications sent to smartphones and tablets and online refers to measuring delay of notifiations sent to devices that are actively being used. It is not certain how much of an impact this 3 second delay will have on caregivers trying to care for at-risk people. More data will have to be collected in an actual hospital environment. Also a push notification fixer application[41] which is used to adjust the GCM default heartbeat interval will need to be used on caregivers' smartphones to be able to have delays like the ones in the online columns even when they are not actively using their phones. The GCM heartbeat is used to keep connections from terminating between the GCM server and smartphone. The GCM heartbeat interval has either two values: 15 minutes for wifi or 28 minutes for cellular. Both are known to be too long causing connections to be dropped and significantly increasing notification delays[42].

The detection times for the standing and sitting-up algorithms are shown in figures 4.1 and 4.2 respectively. It is clear that MOG2 has the best detection times for both region types. The average detection times of MOG2 and Haar differ by about .5 seconds. The average detection times of MOG2 and HOG differ by about 2 seconds. It is difficult to determine how significant these average differences are without collecting data on at-risk people and their process on using the restroom. Some at-risk people may be different in how they use the restroom depending on their ailments, age, or other characteristics.

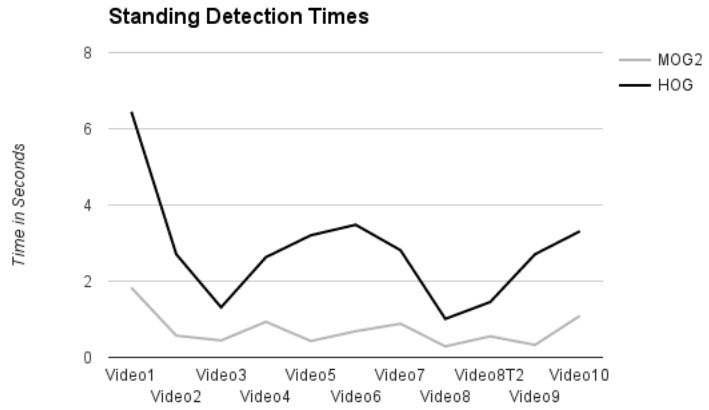

Figure 4.1: A chart of the detection times of the standing algorithms.

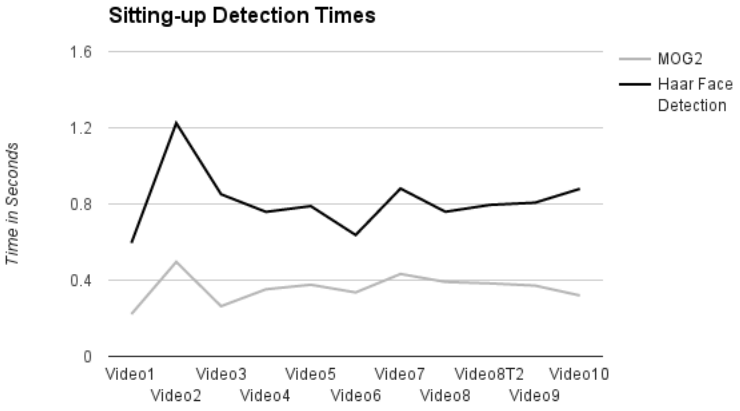

Figure 4.2: A chart of the detection times of the sitting-up algorithms.

Standing		Sitting Up		No Detection
MOG2	HOG	MOG2	HAAR	
7.513	4.934	15.048	12.569	29.324

Table 4.2: The frame rate the smart-camera performs at. Rates are in seconds. "No Detection" is the frame rate without any regions drawn.

Table 4.2 contains the results of the frame rate measurements. The table gives an idea of how each algorithm performs computationally. With the current hardware, it is unlikely that another standing region can be supported, but the hardware may be capable of supporting another sitting-up region. MOG2 is the fastest algorithm for both region types. Last, it is worth mentioning the accuracy of the smart-camera. In our tests, a detection was always made and there were no false positives. There are several possible frames in each test video that the smart-camera can correctly detect, and this is why the detection rate is so high, though in some cases the detection time may be low. The reason behind not having any false positives is that the test videos are mainly static in that their image content does not change, and this content itself does not trigger false positives. This could be an issue in environments that have objects in the detection regions that the detectors falsely identify as a person or face. Also, in standing regions that use HOG for detections, if HOG does not correctly detect a full person but rather an arm, pattern on a shirt, or any other part of a person, this is considered a correct detection if a person is in that standing region at the time. These rates could be different in a hospital environment where IV stands, food trays, or equipment on the wall may trigger detections when none should be made.

5 Conclusion

Our smart camera system provides functionality that would be useful to caregivers who are trying to prevent falls from occurring with at-risk people. Two important functions are provided by our smart camera: generating alerts when an at-risk person is sitting-up and generating alerts when an at-risk person is in the process of standing or is standing. Both alerts are important in fall prevention because they allow a caregiver to know when an at-risk person is in need of assistance walking. When used carefully, the sitting-up alert should only be triggered when a monitored person sits up after sleeping. Because people often need to use the restroom after sleeping, it is assumed a caregiver will be needed to aid the at-risk person to the restroom after waking up. With a caregiver's aid, the chance of falling will decrease. The same is true for the standing alert type. When an at-risk person is in the process of standing up an alert will be sent to a caregiver. The caregiver can come to the aid of the monitored person and guide them to his or her destination. With the help of the caregiver, the chance of falling will go down.

5.1 Future Work

Our smart camera has the computational resources necessary to perform these functionalities. The next stage is to test it in the field. The University of Arkansas for Medical Sciences (UAMS) is a good candidate location for field tests.

Also, our smart camera only operates from a certain perspective. In the future, for use in hospitals, it will need to operate from the perspective behind a patient's bed on the wall. This will be less intrusive for the hospital staff.

Another feature needed is the ability to initially configure the smart camera without having to open the case. One possibility is to use the HostAPD[29] software. This would require a mechanism such as a switch on the outside of the case that would need to be switched to enable hostAPD mode. Once enabled, an administrator could connect to the smart camera that is now acting as a wireless hotspot and make configuration changes. When configuration is done the switch could be turned off.

Lastly, support for detecting more events and objects is needed. Support for detecting when an at-risk person is lying down or how they are lying down could be useful. Also, de-

tecting clutter on the floor could be useful when trying to prevent falls. Detecting a fallen person is important too.

Bibliography

[1] http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0, 2016.

[2] J. Kutchka, D. Tchuinkou, J. M. Mbongue, E. N. Tchinda, and C. Bobda, "Automatic assessment of environmental hazards for fall prevention using smart cameras," in *IEEE Conference on Connected Health: Applications, System Engineering, Technology. Workshop on Cloud Connected Health*, Washington DC, USA, June 27-29, 2016.

[3] Y. S. Yilmaz, B. I. Aydin, and M. Demirbas, "Google cloud messaging (gcm): An evaluation," in *Global Communications Conference (GLOBECOM), IEEE*, Austin, TX, USA, December 8-12, 2014.

[4] "Preventing injuries from patient falls," *American Nurse Today*, vol. 10, no. 7, 2015.

[5] E. L. D. Bouldin, E. M. Andresen, N. E. Dunton, M. Simon, T. M. Waters, M. Liu, M. J. Daniels, L. C. Mion, and R. I. Shorr, "Falls among adult patients hospitalized in the united states," *Journal of Patient Safety*, p. 1, nov 2012.

[6] C. A. Wong, A. J. Recktenwald, M. L. Jones, B. M. Waterman, M. L. Bollini, and W. C. Dunagan, "The cost of serious fall-related injuries at three midwestern hospitals," *The Joint Commission Journal on Quality and Patient Safety*, pp. 81–87, feb 2011.

[7] "Sentinel events alert 55," *The Joint Commission*, sept 2015.

[8] Y. Dionyssiotis, "Analyzing the problem of falls among older people," *International Journal of General Medicine*, p. 805, sep 2012.

[9] L. Z. Rubenstein, "Falls in older people: epidemiology, risk factors and strategies for prevention," *Age and Ageing*, vol. 35, no. Supplement 2, pp. ii37–ii41, sep 2006.

[10] Wikipedia, "C++ — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients.

[11] C. McCormick, "Hog person detector tutorial," http://mccormickml.com/2013/05/09/hog-person-detector-tutorial/, 2013.

[12] ——, "Gradient vectors," http://mccormickml.com/2013/05/07/gradient-vectors/, 2013.

[13] T. Hakim and D. Cohn, "Implementation of hog for human detection," http://www.geocities.ws/talh_davidc/#cst_extract.

[14] http://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html, 2016.

[15] B. Liu, *Web data mining*. Springer, 2011.

[16] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *International Conference on Computer Vision & Pattern Recognition*, C. Schmid, S. Soatto, and C. Tomasi, Eds., vol. 2, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005, pp. 886–893. [Online]. Available: http://lear.inrialpes.fr/pubs/2005/DT05

[17] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, 2001, pp. I–511–I–518 vol.1.

[18] Z. Zivkovic, "Improved adaptive gausian mixture model for background subtraction," in *Proceedings of the International Conference on Pattern Recognition*, 2004.

[19] https://developers.google.com/cloud-messaging/.

[20] M. J. Rantz, T. S. Banerjee, E. Cattoor, S. D. Scott, M. Skubic, and M. Popescu, "Automated fall detection with quality improvement rewind to reduce falls in hospital rooms," *Journal of Gerontological Nursing*, vol. 40, no. 1, pp. 13–17, 2013.

[21] B. Ni, N. C. Dat, and P. Moulin, "Rgbd-camera based get-up event detection for hospital fall prevention," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

[22] http://www.purdue.edu/newsroom/releases/2014/Q2/smartphone-adapted-to-measure-persons-gait,-reduce-falls.html, 2014.

[23] Wikipedia, "C++ — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Asus_EeeBox_PC.

[24] https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[25] http://www.sainsmart.com/raspberry-noir-pi-camera-module-board-5mp-webcam-video-1080p-720p.html.

[26] http://us.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/us/wireless_adapters_n150/ew-7811un/.

[27] J. Brodkin, "Raspberry pi 2 arrives with quad-core cpu, 1gb ram, same 35 price," http://www.google.com, 2010.

[28] Wikipedia, "Raspberry pi — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Raspberry_Pi.

[29] http://w1.fi/.

[30] https://www.amazon.com/Plastic-Security-Shape-Camera-Housing/dp/B00I049JQ4/ref=sr_1_1?m=A1THAZDOWP300U&s=merchant-items&ie=UTF8&qid=1468309158&sr=1-1&keywords=camera+housing.

[31] https://www.raspbian.org/.

[32] http://www.opencv.org.

[33] http://opencv.org/about.html.

[34] Wikipedia, "Qt (software) — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Qt_(software).

[35] https://www.qt.io/licensing/.

[36] https://curl.haxx.se/docs/copyright.html.

[37] https://curl.haxx.se/libcurl/.

[38] http://www.hyperrealm.com/libconfig/.

[39] http://answers.opencv.org/question/74610/would-like-to-use-different-scales-with-hog-people-detection/, 2015.

[40] Wikipedia, "C++ — wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/C%2B%2B.

[41] https://play.google.com/store/apps/details?id=com.andqlimax.pushfixer.noroot&hl=en.

[42] https://productforums.google.com/forum/#!topic/nexus/fslYqYrULto.